Title	Wuの方法の並列化における負荷分散について (数式処理における理論と応用の研究)
Author(s)	白石, 啓一; 那須, 英正; 甲斐, 博; 野田, 松太郎
Citation	数理解析研究所講究録 (2001), 1199: 10-19
Issue Date	2001-04
URL	http://hdl.handle.net/2433/64932
Right	
Type	Departmental Bulletin Paper
Textversion	publisher

数理解析研究所講究録
1199巻 2001年 10-19

Wuの方法の並列化における負荷分散について

詫間電波工業高等専門学校　白石　啓一(Keiichi SHIRAISI) *
愛媛大学 工学部　那須　英正(Hidemasa NASU) †
愛媛大学工学部　甲斐　博(Hiroshi KAI) ‡
愛媛大学工学部　野田　松太郎(Matu-Tarow NODA) §

1 はじめに

連立代数方程式を正確に解くことは、制御・計算機援用設計 (CAD) 等多くの分野において重要である。その手法として、数式処理を用いたGröbner基底計算を基礎にした方法、消去法、characteristic setを計算するWuの方法 [1] を利用する方法などが研究されている。

Gröbner基底計算等が十分に研究され、その高速化がはかられているのに対し、Wuの方法に対する考察やインプリメントは十分ではない。そこで、我々は、並列処理の利用による高速化を考え、並列計算機AP3000へ実装した [2]。Wuの方法では、それぞれ独立に計算できる多項式の擬除算を繰り返し計算するため、それらを複数のプロセッサに割り当てることにより、容易に並列化できる。しかし、単純に擬除算の回数を平等に分けただけでは、計算負荷が平均化されず、高負荷がかかっているプロセッサの計算時間により、全体の計算時間が束縛される。そこで、本稿では、Wuの方法の並列化における負荷分散の方法とその一部についての実装を報告する。

以下、2節ではWuの方法について、3節ではWuの方法の並列化と負荷分散について述べる。4節では3節で述べた負荷分散の実装と実験結果を述べる。最後に、5節で結果と今後の課題をまとめる。

*siraisi@dc.takuma-ct.ac.jp
†h_nasu@hpc.cs.ehime-u.ac.jp
‡kai@cs.ehime-u.ac.jp
§noda@cs.ehime-u.ac.jp

2 Wu の方法

Wu の方法とは、多項式集合 PS の characteristic set CS を計算する方法である。CS は、L-zero$(\cdot)$ で多項式集合の零点を表すと、

$$L\text{-zero}(CS/J) \subseteq L\text{-zero}(PS) \subseteq L\text{-zero}(CS)$$

を満たす。ここで、$J = \prod_{f \in CS} \text{ini}(f)$ であり、ini(f) は多項式 f の主係数を表す。

Wu の方法のアルゴリズムを示す前に、用語を簡単に説明する。Wu の方法では、多変数多項式集合を扱う。この多変数多項式に含まれる変数には順序が決められている。以後、

$$x_1 \prec x_2 \prec \cdots \prec x_n$$

と考える。ある多項式に含まれる変数のうち、順序最大 (添数最大) の変数を主変数と呼び、lvar(f) で表す。

非零多項式の有限個の列 $P_1, P_2, \ldots, P_r$ が、次の 2 つの条件のうち、どちらか一方を満たしているとき、この多項式集合を ascending set と呼ぶ。

1. $r = 1$ かつ P_1 は定数
2. lvar$(P_1) \prec$ lvar$(P_2) \prec \cdots \prec$ lvar(P_r) 、かつ、整数の対 $(i, j)(0 < i < j \leq r)$ それぞれについて、多項式 P_i,P_j が次の条件を満たす
   P_i の主変数を x_i とすると、

$$\deg_{x_i} P_i > \deg_{x_i} P_j$$

ここで、$\deg_x f$ は多項式 f の変数 x に関する最大次数を表す。

環 $k[x_1, x_2, \ldots, x_n, y]$ 上の 2 つの多項式

$$\begin{aligned} f &= c_p y^p + \cdots + c_1 y + c_0 \\ g &= d_m y^m + \cdots + d_1 y + d_0 \end{aligned}$$

を考える。多項式 f の y に関する多項式 g による擬剰余 r は、

$$d_m^s \cdot f = q \cdot g + r$$

を満たすように決められ、$r = \text{prem}(f, g, y)$ と表す。q は擬商と呼ばれる。また、多項式集合 $PS = \{P_1, P_2, \ldots, P_n\}$ が ascending set ならば、多項式 f の多項式集合 PS による擬

剰余 r_0 は以下の計算で求められる。

$$\begin{aligned} r_{n-1} &= \text{prem}(f, P_n, \text{lvar}(P_n)) \\ r_{n-2} &= \text{prem}(r_{n-1}, P_{n-1}, \text{lvar}(P_{n-1})) \\ &\vdots \\ r_1 &= \text{prem}(r_2, P_2, \text{lvar}(P_2)) \\ r_0 &= \text{prem}(r_1, P_1, \text{lvar}(P_1)) \end{aligned}$$

$r_0 = \text{premas}(f, PS)$ と表す。

CS を求める Wu の方法のアルゴリズムは以下のように記述される。

アルゴリズム 1 (Wu の方法)
入力 : 多項式集合 $PS = \{PS_1, PS_2, \ldots, PS_n\}$
出力 : *characteristic set* CS

1. PS より *ascending set* の条件を満たすように多項式集合 *(*PS の部分集合*)* を選択し、CS とする。

2. $RS = \emptyset$
   $i = 1, 2, \ldots, n$ について、

   $$RS = RS \cup premas(PS_i, CS)$$

   を計算する。

3. $RS = \emptyset$ ならば、終了。$RS \neq \emptyset$ ならば、$PS = PS \cup RS$ とし、*1.* へ。

3 Wu の方法の並列化と負荷分散

2 節で示した Wu の方法のアルゴリズムの手順 2 では、擬剰余演算 $\text{premas}(PS_i, CS)$ をそれぞれ独立に行うことができる。また、Wu の方法のアルゴリズムの中で、手順 2 に最も計算時間がかかる。そこで、我々はこれら擬剰余演算を複数のプロセッサに分散させて並列に処理させた [2]。並列計算のモデルとしてマスタ–ワーカ・モデルを用い、マスタでは計算しないこととした (図 1)。ワーカへの擬剰余演算の割当は、擬剰余演算の回数が各プロセッサで平均化されるように分散させた。即ち、擬剰余演算回数を n 、プロセッサ数を m としたとき、各プロセッサへ n/m 回の擬剰余演算を割り当てた。すると、あるプロセッサの CPU time が他に較べ大きくなる現象が観察された。これは負荷の不均衡のために起こっている。Wu の方法では、次の理由で負荷の不均衡が起こる [3]。

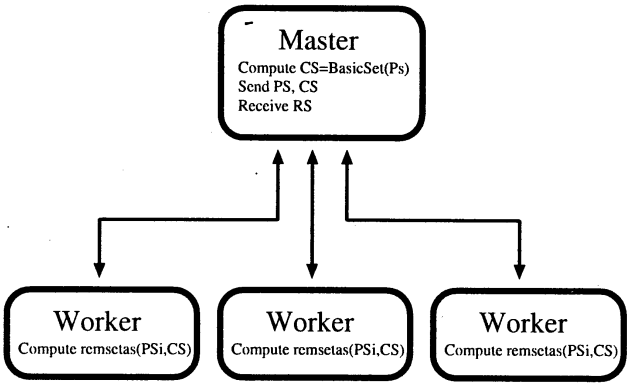

図 1: マスター–ワーカ・モデル

- ワーカの数より擬剰余演算回数が少ない場合、いくつかのワーカはアイドル状態になる。この状況は問題が小さい場合に起こり、並列処理はそれほど効果的でない。
- 多項式が異なると、擬剰余演算に要する計算時間も変化する。そのため、ワーカがマスタから同じ数の多項式を受け取った場合、いくつかのワーカでは擬剰余演算を終えていても、他のワーカでは終えていない。

パフォーマンスを良くするためには、負荷分散を行わねばならない。

ジョブの静的割当、動的割当　各ジョブをワーカへ割り当てる際、ある規則に従って割り当てる方法を静的割当と呼び、各ワーカの負荷を考慮しながらジョブを割り当てる方法を動的割当と呼ぶ。Wu の方法では、ジョブの割当が手順 2 で行われる。静的割当では、手順 2 を始める時点で全てのジョブ (擬剰余演算 premas(PS_i, CS)) を各ワーカへ割り当てる。動的割当では、手順 2 の始めに各ワーカへ 1 個づつジョブを割り当て、その後、計算を終えたワーカへ順にジョブを割り当てる。

処理すべきジョブの実行時間が既知であると仮定する。実行時間が既知であることより、最適なスケジュールを作ることができる。このスケジュールは静的割当、動的割当にかかわらず等しい。従って、通信時間を無視して、処理時間は変わらない。静的割当と動的割当の違いは通信コストにあり、通常、動的割当での通信コストが高い。

しかし、実際の問題では正確な実行時間の予測が困難なので、静的割当によりジョブを割り当てていると負荷の不均衡が起こる。動的割当ならば、仕事を終えたプロセッサに次の仕事を割り当てることで負荷を分散できる。従って、負荷分散を行うためには、動的割当が有利である。

故意の休止を含まないスケジュール　最適なスケジュールを作るための (徹底した試行錯誤よりも短い) アルゴリズムは、未だ知られていないが、故意の休止を含まないスケジュール

を立てることで比較的良いスケジュールを作成できることが知られている [4]。故意の休止を含まないスケジュールでは、実行可能なジョブがないときのみ、プロセッサを休ませる。

例として、図 2 に示す作業を考える。"/" の左に書いてある文字がジョブの名前、右に書いてある数字がそのジョブの実行時間である。有向辺の終点にあたるジョブを始めるには、始点にある仕事を終えなければならない。例えば、T4 を始めるために、T1 と T2 を終えなければならない。

この作業を 2 個のプロセッサを使って行う場合、図 3 左のスケジュールが最適である。ここで、休止を ϕ で表している。このスケジュールでは、T2 を終えた後、T3 が実行可能であるにもかかわらず、1 の休止を入れることにより、全経過時間を最小にしている。

図 3 右のスケジュールが故意の休止を含まないスケジュールである。このスケジュールでは、T2 を終えた後、実行可能な T3 を実行している。その後、実行可能なジョブを順に実行している。

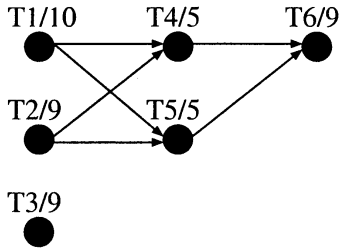

図 2: 作業工程図 (例)

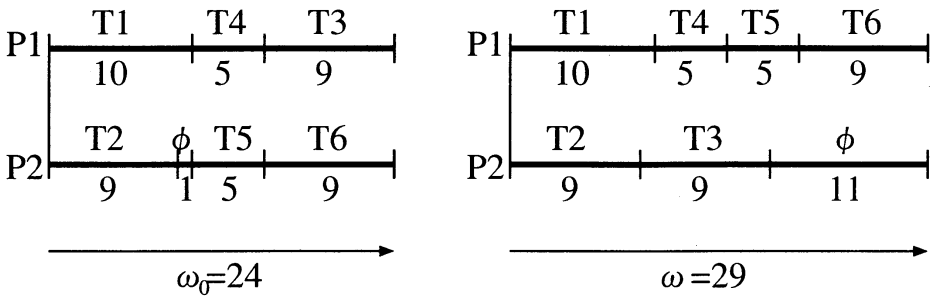

図 3: スケジュール (左:故意の休止を含む、右:故意の休止を含まない)

"故意の休止を含まないスケジュール" がどの程度良いものかを示す次の定理がある [4]。

定理 2 与えられたジョブの集合に対して、ジョブが故意の休止時間を含まないあるスケジュールに従って実行されるときの全経過時間を ω で表し、可能な最小全経過時間を ω_0 で表す。このとき、

$$\frac{w}{w_0} \leq 2 - \frac{1}{n}$$

となる。ここで、n はその計算システムにおけるプロセッサの台数である。さらに、この上界は最良である。

故意の休止を含まないスケジュールを立てることにより、全経過時間は最小全経過時間の 2 倍で押えられると保証されたので、この中で、できるだけ全経過時間を短くすることを考える。

ジョブ割当の順序　ここでは、依存関係のないジョブの割当順を考える。即ち、全てのジョブは任意の順序で実行可能である。 Wu の方法では、図 4 に示す通り、手順 2 の premas(PS_i, CS) の割当順にあたる。ここで、 1,3 は手順 1,3 を、 2-1,2-2,...,2-n はそれぞれ手順 2 の premas(PS_i, CS) を表している。

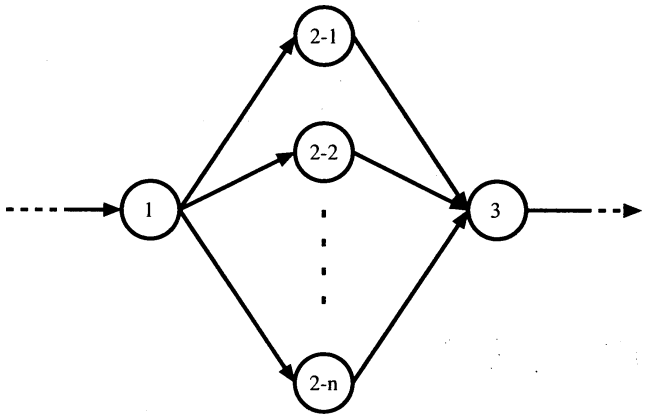

図 4: 作業工程図 (Wu の方法)

まず、ジョブの実行時間が既知であり、n 個のジョブがあると仮定する。$n-1$ 個のジョブがプロセッサにほぼ均等に割り当てられているとき、最後の 1 個のジョブを処理時間が最短であるプロセッサへ割り当てることで全ジョブを終えることができる。このとき、実行時間が最小のジョブを割り当てると良いスケジュールを立てることができる。そこで、次のアルゴリズムを考える。

アルゴリズム 3 (ジョブ割当)
入力 : ジョブの集合 $T = \{T_1, T_2, \ldots, T_n\}$, ジョブの実行時間 $\mu(T_i)$, プロセッサ数 m
出力 : スケジュール

1. 実行時間が最長のジョブを処理時間が最短のプロセッサに割り当て、そのジョブを T から除く。(処理時間が最短のプロセッサが複数あれば、どのプロセッサに割り当てても良い。)

2. $T = \emptyset$ ならば終了。そうでなければ、 1. へ。

ただし、アルゴリズム 2 “ジョブ割当” は最適なスケジュールを与えない。例えば、ジョブ集合 $T = \{T_1/3, T_2/3, T_3/2, T_4/2, T_5/2\}$ を 2 個のプロセッサ P_1, P_2 へ割り当てることを考える。アルゴリズム 2 を用いると、 P_1 へ T_1, T_3, T_5 を、P_2 へ T_2, T_4 を割り当て、処理時間はそれぞれ 7, 5 になる。しかし、明らかに最適なスケジュールは P_1 へ T_1, T_2 を、P_2 へ T_3, T_4, T_5 を割り当てることである。

次に、ジョブの実行時間が未知の場合を考える。この場合、予測実行時間を元にアルゴリズム 2 を実行する。静的割当を行った場合、負荷の不均衡が大きくなると考えられる。動的割当を行えば、予測実行時間より短い時間で処理が終ったときにそのプロセッサに新しいジョブを割り当て、予測実行時間よりも長い時間がかかったときに他のプロセッサにジョブを割り当てることで、比較的良いスケジュールが得られる。

以上より、ジョブを予測実行時間の長い順に動的に割り当てれば、通信コストと実行時間予測コストを無視して、良いスケジュールが得られる。また、動的割当を行えば、実行時間予測をしなくとも、定理 1 より最小全経過時間の 2 倍未満の時間で実行できる。通信コストを考慮すると、最初にジョブの一部をまとめて割り当てておき、残ったジョブを動的に割り当てる方法が良いが、その最初に割り当てるジョブの割合は実験的に検証する必要がある [3]。

4 負荷分散の実装と実験

3 節で述べたように、負荷分散には大きく分けて次の 5 種類の方法がある。

1. (予測) 実行時間を考慮した動的割当
2. 動的割当
3. (予測) 実行時間を考慮した静的割当
4. 静的割当
5. ハイブリッド (静的割当と動的割当を組み合わせた方法)

今回、我々は、 2 の動的割当を用いた負荷分散を、並列化した Wu の方法へ組み込み、数式処理システム Risa/Asir 上で実装した。まず、そのアルゴリズムをマスタとワーカに分けて、述べる。

アルゴリズム 4 (Wu の方法 (マスタ))
入力 : 多項式集合 $PS = \{PS_1, PS_2, \ldots, PS_n\}$
出力 : *characteristic set* CS

1. PS より *ascending set* の条件を満たすように多項式集合 *(PS の部分集合)* を選択し、CS とする。

2. $RS = \emptyset$

3. アイドル状態のワーカへ PS_i, CS を送り、 PS_i を PS から除く。

4. 計算を終えたワーカより結果 RS_i を受け取り、

$$RS = RS \cup premas(PS_i, CS)$$

を計算する。

5. $PS \neq \emptyset$ ならば、 *3.* へ。

6. $RS = \emptyset$ ならば、終了。$RS \neq \emptyset$ ならば、 $PS = PS \cup RS$ とし、 *1.* へ。

アルゴリズム 5 (Wu の方法 (ワーカ))

1. マスタより PS_i, CS を受け取る。

2. $RS_i = premas(PS_i, CS)$ を計算する。

3. マスタへ RS_i を送る。

ジョブの送信、結果の受信には Risa/Asir の ox_rpc() 関数、ox_get() 関数を用いた。また、Risa/Asir の通信は TCP/IP により行われている。

例題として次の問題を挙げる。

$$\begin{aligned} PS \quad = \quad & \{y^2z + 2xyt - 2x - z, -x^3z + 4xy^2z + 4x^2yt \\ & +2y^3t + 4x^2 - 10y^2 + 4xz - 10yt + 2, \\ & 2yzt + xt^2 - x - 2z, -xz^3 + 4yz^2t + 4xzt^2 \\ & +2yt^3 + 4xz + 4z^2 - 10yt - 10t^2 + 2\} \end{aligned}$$

$$t \prec x \prec y \prec z$$

計算は AP3000(ノード: UltraSPARCII 360MHz×2, メモリ 640MB, 6 ノード/ 1 パーティション) 上で行い、並列計算ではプロセッサ数を 2,4,6 へ変化させた。計算時間は Risa/Asir が持つ time() 関数で計り、実時間により比較した (表 1)。なお、表中の並列版でのプロセッサ数には、マスタ、ワーカともに含めているので、実際に計算を行うプロセッサ (ワーカ) は 1 減る。実験結果より、

- プロセッサ数を増やすと逐次計算よりも並列計算が高速になること

表 1: 計算時間 (単位: 秒)

		プロセッサ数	計算時間	並列度 (T_S/T_P)
逐次処理	T_S	1	53.1	—
並列処理	T_{P2}	2	103.8	0.51
	T_{P4}	4	37.14	1.43
	T_{P6}	6	24.0	2.21

- 並列計算はプロッサ数を増やす程、高速になること

が分かる。プロセッサ数が 2 の場合、逐次処理より並列処理が遅いのは、ワーカが 1 であるため、擬剰余演算部が並列に計算されておらず、通信時間が余分にかかっているためである。特に、動的割当ではジョブ (ここでは擬剰余演算) を 1 個ずつ送っているために通信回数が増えるため、通信時間が長くなる。例題では 589 回の送受信が行われている。同じ例題を静的割当によって計算した場合、プロセッサ数 2 のとき、送受信回数は 13 回、計算時間は 54.9 秒である [2]。ただし、負荷分散は期待できない。負荷分散と通信のコストを考慮すると、一部のジョブをまとめて割り当てておき、残りのジョブを動的に割り当てる必要がある。

5 おわりに

characteristic set を求めるための Wu の方法を並列化し、負荷分散について考察し、

- ジョブの静的割当、動的割当
- 故意の休止を含まないスケジュール
- ジョブ割当の順序

についてまとめた。また、本稿では、動的割当による負荷分散について実装を行い、実際の計算例を通して、並列化による高速性を示すことが出来た。しかし、通信回数が増えることにより通信時間増大という問題が起きることが分かった。これは、分散させる計算の粒度が小さすぎるとも考えられる。今後の課題として、その他の負荷分散法の実装し、比較検討を行うことが挙げられる。また、実際の工学問題に応用するには、浮動小数係数の多項式を扱えるようにする必要がある。数値数式融合アルゴリズムを取り入れていきたい。

参 考 文 献

[1] Wu Wen-tsün: A Mechanization Method of Equations-solving and Theorem-proving, *Advances in Computing Research*, **6**, 1992, 103–138.

[2] 白石 啓一, 那須 英正, 野田 松太郎: Characteristic Set を求めるための Wu の方法の並列化について, 情報処理学会第 61 回 (平成 12 年度後期) 全国大会, 情報処理学会, 東京, 2000, 1-151–1-152.

[3] Iyad A. Ajwa: Parallel Algorithms and Implementations for the Gröbner Bases Algorithm and the Characteristic Sets Method, Ph.D. Dissertation, Kent State University, Kent, 1998.

[4] C. L. Liu: *Elements of Discrete Mathematics, 2nd Edition*, McGraw-Hill, 1985. (邦訳: C. L. リュー: 離散数学入門 (成嶋 弘, 秋山 仁 訳), マグロウヒル出版, 東京, 1986)